On the Computational Complexity of MapReduce*

Benjamin Fish†, Jeremy Kun, Ádám D. Lelkes, Lev Reyzin and György Turán

Department of Mathematics, Statistics, and Computer Science
University of Illinois at Chicago
Chicago, IL 60607
{bfish3,jkun2,alelke2,lreyzin,gyt}@uic.edu

Abstract

In this paper we study the MRC model, which aims to formally capture distributed MapReduce computations. We show that the class of regular languages, and moreover all of sublogarithmic space, lies in constant-round MRC. In addition, we prove that, conditioned on a weak version of the Exponential Time Hypothesis, there are strict hierarchies within MRC so that increasing the number of rounds or the amount of time per processor increases the power of MRC. Our work lays the foundation for further analysis relating MapReduce to established complexity classes. Our results also hold for Valiant's BSP model of parallel computation and the MPC model of Beame et al.

*Regular submission. Benjamin Fish, Jeremy Kun, and Ádám D. Lelkes are full-time students. This paper is eligible for the best student paper award.

†Contact author. Address: 851 S. Morgan St. Chicago, IL 60607. Phone: (312)-996-3041.

1 Introduction

MapReduce is a programming model originally developed to separate algorithm design from the engineering challenges of massively distributed computing. A programmer can separately implement a "map" function and a "reduce" function that satisfy certain constraints, and the underlying MapReduce technology handles all the communication, load balancing, fault tolerance, and scaling. MapReduce frameworks and their variants have been successfully deployed in industry by Google [4], Yahoo! [18], and many others.

MapReduce offers a unique and novel model of parallel computation because it alternates parallel and sequential steps, and imposes sharp constraints on communication and random access to the data. This distinguishes MapReduce from classical theoretical models of parallel computation and this, along with its popoularity in industry, is a strong motivation to study the theoretical power of MapReduce. From a theoretical standpoint we ask how MapReduce relates to established complexity classes. From a practical standpoint we ask which problems can be efficiently modeled using MapReduce and which cannot.

In 2010 Karloff et al. [12] initiated a principled theoretical study of MapReduce, providing the definition of the complexity class MRC and comparing it with the classical PRAM models of parallel computing. But to our knowledge, since this initial paper, almost all of the work on MapReduce has focused on algorithmic issues.

Complexity theory studies the classes of problems defined by resource bounds on different models of computation. A central goal of complexity theory is to understand the relationships between different models, i.e. to see if the problems solvable with bounded resources on one computational model can be solved with a related resource bound on a different model. In this paper we prove a result that establishes a connection between MapReduce and space-bounded computation on classical Turing machines. Another traditional question asked by complexity theory is whether increasing the resource bound on a certain computational resource strictly increases the set of solvable problems. Such so-called hierarchy theorems exist for time and space on deterministic and non-deterministic Turing machines, among other settings. In this paper we prove conditional hierarchy theorems for MapReduce rounds and time.

First we lay a more precise theoretical foundation for studying MapReduce computations (Section 3). In particular, we observe that Karloff et al.'s definitions are non-uniform, allowing the complexity class to contain undecidable languages. We reformulate the definition of [12] to make a uniform model and to more finely track the parameters involved (Section 3.2). In addition, we point out that our results hold for other important models of parallel computations, including BSP and MPC (Section 3.3). We then prove two main theorems: SPACE($o(\log n)$) has constant-round MapReduce computations (Section 4) and, conditioned on a version of the Exponential Time Hypothesis, there are strict hierarchies within MRC. In particular, sufficiently increasing time or number of rounds increases the power of MRC (Section 5).

Our sub-logarithmic space result is achieved by a direct simulation, using a two-round protocol that localizes state-to-state transitions to the section of the input being simulated, combining the sections in the second round. Our hierarchy theorem involves proving a (conditional) time hierarchy within linear spaceachieved by a padding argument, along with proving a time-and-space upper and lower bounds on simulating MRC machines within P. To the best of our knowledge our hierarchy theorem is the first of its kind. We conclude with a discussion and open questions raised by our work (Section 6).

2 Background and previous work

2.1 MapReduce

The MapReduce protocol can be roughly described as follows. The input data is given as a list of key-value pairs, and over a series of rounds two things happen per round: a "mapper" is applied to each key-value pair independently (in parallel), and then for each distinct key a "reducer" is applied to all corresponding values for a group of keys. The canonical example is counting word frequencies with a two-round MapReduce protocol. The inputs are (index, word) pairs, the first mapper maps $(k, v) \mapsto (v, k)$, and the first reducer computes the sum of the word frequencies for the given key. In the second round the mapper sends all data to a single processor via $(k, n_k) \mapsto (1, (k, n_k))$, and the second processor formats the output appropriately.

One of the primary challenges in MapReduce is data locality. MapReduce was designed for processing massive data sets, so MapReduce programs require that every reducer only has access to a substantially sublinear portion of the input, and the strict modularization prohibits reducers from communicating within a round. All communication happens indirectly through mappers, which are limited in power by the independence requirement. Finally, it's understood in practice that a critical quantity to optimize for is the number of rounds [12], so algorithms which cannot avoid a large number of rounds are considered inefficient and unsuitable for MapReduce.

There are a number of MapReduce-like models in the literature, including the MRC model of Karloff et al. [12], the "mud" algorithms of Feldman et al. [6], Valiant's BSP model [20], the MPC model of Beame et al. [2], and extensions or generalizations of these, e.g. [8]. The MRC class of Karloff et al. is the closest to existing MapReduce computations, and is also among the most restrictive in terms of how it handles communication and tracks the computational power of individual processors. In their influential paper [12], Karloff et al. display the algorithmic power of MRC, and prove that MapReduce algorithms can simulate CREW PRAMs which use subquadratic total memory and processors.

Since [12], there has been extensive work in developing efficient algorithms in MapReduce-like frameworks. For example, Kumar et al. [13] analyze a sampling technique allowing them to translate sequential greedy algorithms into log-round MapReduce algorithms with a small loss of quality. Farahat et al. [5] investigate the potential for sparsifying distributed data using random projections. Kamara and Raykova [11] develop a homomorphic encryption scheme for MapReduce. And much work has been done on graph problems such as connectivity, matchings, sorting, and searching [8]. Chu et al. [3] demonstrate the potential to express any statistical-query learning algorithm in MapReduce. Finally, Sarma et al. [16] explore the relationship between communication costs and the degree to which a computation is parallel in one-round MapReduce problems. Many of these papers pose general upper and lower bounds on MapReduce computations as an open problem, and to the best of our knowledge our results are the first to do so with classical complexity classes.

The study of MapReduce has resulted in a wealth of new and novel algorithms, many of which run faster than their counterparts in classical PRAM models. As such, a more detailed study of the theoretical power of MapReduce is warranted. Our paper contributes to this by establishing a more precise definition of the MapReduce complexity class, proving that it contains sublogarithmic deterministic space, and showing the existence of certain kinds of hierarchies.

2.2 Complexity

From a complexity-theory viewpoint, MapReduce is unique in that it combines bounds on time, space and communication. Each of these bounds would be very weak on its own: the total time available to processors is polynomial; the total space and communication are slightly less than quadratic. In particular, even though arranging the communication between processors is one of the most difficult parts of designing a MapReduce algorithms, classical results from communication complexity do not apply since the total communication available is more than linear. These innocent-looking bounds lead to serious restrictions when combined, as demonstrated by the fact that it is unknown whether constant-round MRC machines can decide graph connectivity (the best known result achieves a logarithmic number of rounds with high probability [12]), although it is solvable using only logarithmic space on a deterministic Turing machine.

We relate the MRC model to more classical complexity classes by studying simultaneous time-space bounds. $\mathrm{TISP}(T(n), S(n))$ are the problems that can be decided by a Turing machine which on inputs of length n takes at most $O(T(n))$ time and uses at most $O(S(n))$ space. Note that in general it is believed that $\mathrm{TISP}(T(n), S(n)) \neq \mathrm{TIME}(T(n)) \cap \mathrm{SPACE}(S(n))$. The complexity class TISP is studied in the context of time-space tradeoffs (see, for example, [7, 22]). Unfortunately much less is known about TISP than about TIME or SPACE; for example there is no known time hierarchy theorem for fixed space. The existence of such a hierarchy is mentioned as a problem in the monograph of Wagner and Wechsung [21].

To prove the results about TISP that imply the existence of a hierarchy in MRC, we use the Exponential Time Hypothesis (ETH) introduced by Impagliazzo, Paturi, and Zane [9, 10], which conjectures that 3-SAT is not in $\mathrm{TIME}(2^{cn})$ for some $c > 0$. This hypothesis and its strong version have been used to prove conditional lower bounds for specific hard problems like vertex cover, and for algorithms in the context of fixed parameter tractability (see, e.g., the survey of Lokshtanov, Marx and Saurabh [14]). The first open problem mentioned in [14] is to relate ETH to some other known complexity theoretic hypotheses.

We show in Lemma 5 that ETH implies directly a time-space trade-off statement involving time-space complexity classes. This statement is not a well-known complexity theoretic hypothesis, although it is related to the existence of a time hierarchy with a fixed space bound. In fact, as detailed in Section 5, a hypothesis weaker than ETH is sufficient for the lemma. The relative strengths of ETH, the weaker hypothesis, and the statement of the lemma seem to be unknown.

3 Models

In this section we introduce the model we will use in this paper, a uniform version of Karloff's MapReduce Class (MRC), and contrast MRC to other models of parallel computation, such as Valiant's Bulk-Synchronous Parallel (BSP) model, for which our results also hold.

3.1 MapReduce and MRC

The central piece of data in MRC is the key-value pair, which we denote by a pair of strings $\langle k, v \rangle$, where k is the key and v is the value. An input to an MRC machine is a list of key-value pairs $\langle k_i, v_i \rangle_{i=1}^{N}$ with a total size of $n = \sum_{i=1}^{N} |k_i| + |v_i|$. The definitions in this subsection are adapted from [12].

Definition 1. A *mapper* μ is a Turing machine[1] which accepts as input a single key-value pair $\langle k, v\rangle$ and produces a list of key-value pairs $\langle k'_1, v'_1\rangle, \ldots, \langle k'_s, v'_s\rangle$.

Definition 2. A *reducer* ρ is a Turing machine which accepts as input a key k and a list of values $\langle v_1, \ldots, v_m\rangle$, and produces as output the same key and a new list of values $\langle v'_1, \ldots, v'_M\rangle$.

Definition 3. For a decision problem, an input string $x \in \{0,1\}^*$ to an MRC machine is the list of pairs $\langle i, x_i\rangle_{i=1}^n$ describing the index and value of each bit. We will denote by $\langle x\rangle$ the list $\langle i, x_i\rangle$.

An MRC machine operates in rounds. In each round, a set of mappers running in parallel first process all the key-value pairs. Then the pairs are partitioned (by a mechanism called "shuffle and sort" that is not considered part of the runtime of an MRC machine) so that each reducer only receives key-value pairs for a single key. Then the reducers process their data in parallel, and the results are merged to form the list of key-value pairs for the next round. More formally:

Definition 4. An *R-round MRC machine* is an alternating list of mappers and reducers $M = (\mu_1, \rho_1, \ldots, \mu_R, \rho_R)$. The execution of the machine is as follows. For each $r = 1, \ldots, R$:

1. Let U_{r-1} be the list of key-value pairs processed from the last round (or the input pairs when $r = 1$). Apply μ_r to each key-value pair of U_{r-1} to get the multiset $V_r = \bigcup_{\langle k,v\rangle \in U_{r-1}} \mu_r(k, v)$.
2. Shuffle-and-sort groups the values by key. Call each piece $V_{k,r} = \{k, (v_{k,1}, \ldots, v_{k,s_k})\}$.
3. Assign a different copy of reducer ρ_r to each $V_{k,r}$ (run in parallel) and set $U_r = \bigcup_k \rho_r(V_{k,r})$.

The output is the final set of key-value pairs. For decision problems, we define M to accept $\langle x\rangle$ if in the final round $U_R = \emptyset$. Equivalently we may give each reducer a special accept state and say the machine accepts if at any time any reducer enters the accept state. We say M *decides* a language L if it accepts $\langle x\rangle$ if and only if $x \in L$.

The central caveat that makes MRC an interesting class is that the reducers have space constraints that are sublinear in the size of the input string. In other words, no sequential computation may happen that has random access to the entire input. Thinking of the reducers as processors, cooperation between reducers is obtained not by message passing or shared memory, but rather across rounds in which there is a global communication step.

In the MRC model we use in this paper, we require that every mapper and reducer arise as separate runs of the same Turing machine M. Our Turing machine $M(m, r, n, y)$ will accept as input the current round number r, a bit m denoting whether to run the r-th map or reduce function, the total number of rounds n, and the corresponding input y. Equivalently, we can imagine a list of mappers and reducers in each round $\mu_1, \rho_1, \mu_2, \rho_2, \ldots$, where the descriptions of the μ_i, ρ_i are computable in polynomial time in $|i|$.

Definition 5 (Uniform Deterministic MRC)**.** A language L is said to be in MRC$[f(n), g(n)]$ if there is a constant $0 < c < 1$, an $O(n^c)$-space and $O(g(n))$-time Turing machine $M(m, r, n, y)$, and an $R = O(f(n))$, such that for all $x \in \{0,1\}^n$, the following holds.

1. Letting $\mu_r = M(1, r, n, -), \rho_r = M(0, r, n, -)$, the MRC machine $M_R = (\mu_1, \rho_1, \ldots, \mu_R, \rho_R)$ accepts x if and only if $x \in L$.

[1]The definitions of [12] were for RAMs. However, because we wish to relate MapReduce to classical complexity classes, we reformulate the definitions here in terms of Turing machines.

2. Each μ_r outputs $O(n^c)$ distinct keys.

This definition closely hews to practical MapReduce computations: $f(n)$ represents the number of times global communication has to be performed, $g(n)$ represents the time each processor gets, and space bounds in terms of $n = |x|$ ensure that the size of the data on each processor is smaller than the full input.

Remark 1. By $M(1, r, n, -)$, we mean that the tape of M is initialized by the string $\langle 1, r, n\rangle$. In particular, this prohibits an MRC algorithm from having $2^{\Omega(n)}$ rounds; the space constraints would prohibit it from storing the round number.

Remark 2. Note that a polynomial time Turing machine with sufficient time can trivially simulate a uniform MRC machine. All that is required is for the machine to perform the key grouping manually, and run the MRC machine as a subroutine. As such, $\mathrm{MRC}[\mathrm{poly}(n), \mathrm{poly}(n)] \subseteq P$. We give a more precise computation of the amount of overhead required in the proof of Lemma 7.

Definition 6. Define by MRC^i the union of uniform MRC classes

$$\mathrm{MRC}^i = \bigcup_{k\in\mathbb{N}} \mathrm{MRC}[\log^i(n), n^k].$$

So in particular $\mathrm{MRC}^0 = \bigcup_{k\in\mathbb{N}} \mathrm{MRC}[1, n^k]$.

3.2 Nonuniformity

A complexity class is generally called uniform if the descriptions of the machines solving problems in it do not depend on the input length. Classical complexity classes defined by Turing machines with resource bounds, such as P, NP, and SPACE($\log(n)$), are uniform. On the other hand, circuit complexity classes are naturally nonuniform since a fixed Boolean circuit can only accept inputs of a single length. There is ambiguity about the uniformity of MRC as defined in [12]. Since we wish to relate the MRC model to classical complexity classes such as P and SPACE($\log(n)$), making sure that the model is uniform is crucial. Indeed, innocuous-seeming changes to the definitions above introduce nonuniformity (and in particular this is true of the original MRC definition in [12]). In the appendix we show that the nonuniform MRC model defined in [12] allows MRC machines to solve undecidable problems in a logarithmic number of rounds, including the halting problem. We introduced our uniform version of MRC above to rule out such pathological behavior.

3.3 Other models of parallel computation

Several other models of parallel computation have been introduced, including the BSP model of Valiant [20] and the MPC model of Beame et. al. [2]. The main difference between BSP and MapReduce is that in the BSP models the key-value pairs and the shuffling steps needed to redistribute them are replaced with point-to-point messages. Similarly to [12], in Valiant's paper [20] there is also ambiguity about the uniformity of the model. In this paper, when we refer to BSP we mean a uniform deterministic version of the model. We give the exact definition in the appendix.

Goodrich et al. [8] and Pace [15] showed that MapReduce computations can be simulated in the BSP model and vice versa, with only a constant blow-up in the computational resources needed. This implies that our theorems about MapReduce automatically apply to BSP.

Similarly, the MPC model uses point-to-point messages and Beame et. al.'s paper [2] does not discuss the uniformity of the model. The main distinguishing charateristic of the MPC model is that it introduces the number of processors p as an explicit paramter. Setting $p = O(n^c)$, our results will also hold in this model.

There are other variants of these models, including the model that Andoni et. al. [1] uses, which follows the MPC model but also introduces the additional constraint that total space used across each round must be no more than $O(n)$. It is straightforward to check that the proofs of our results never use more than $O(n)$ space, implying that our results hold even under this more restrictive model.

4 Space complexity classes in MRC^0

In this section we prove that small space classes are contained in constant-round MRC. Again, the results in this section also hold for other similar models of parallel computation, including the BSP model and the MPC model. First, we prove that the class REGULAR of regular languages is in MRC^0. It is well known that SPACE($O(1)$) = REGULAR [17], and so this result can be viewed as a warm-up to the theorem that SPACE($o(\log n)$) $\subseteq$ MRC^0. Indeed, both proofs share the same flavor, which we sketch before proceeding to the details.

In the first round each parallel processor receives a contiguous portion of the input string and constructs a state transition function using the data of the globally known DFA. Though only the processor with the beginning of the string knows the true state of the machine during its portion of the input, all processors can still compute the *entire* table of state-to-state transitions for the given portion of input. In the second round, one processor collects the transition tables and chains together the computations, and this step requires only the first bit of input and the list of tables.

We can count up the space and time requirements to prove the following theorem.

Theorem 1. REGULAR $\subsetneq$ MRC^0

Proof. Let L be a regular language and D a deterministic finite automaton recognizing L. Define the first mapper so that the j^{th} processor has the bits from $j\sqrt{n}$ to $(j+1)\sqrt{n}$. This means we have $K = O(\sqrt{n})$ processors in the first round. Because the description of D is independent of the size of the input string, we also assume each processor has access to the relevant set of states S and the transition function $t : S \times \{0, 1\} \to S$.

We now define ρ_1. Fix a processor j and call its portion of the input y. The processor constructs a table T_j of size at most $|S|^2 = O(1)$ by simulating D on y starting from all possible states and recording the state at the end of the simulation. It then passes T_j and the first bit of y to the single processor in the second round.

In the second round the sole processor has K tables T_j and the first bit x_1 of the input string x (among others but these are ignored). Treating T_j as a function, this processor computes $q = T_K(\ldots T_2(T_1(x_1)))$ and accepts if and only if q is an accepting state. This requires $O(\sqrt{n})$ space and time and proves containment. To show this is strict, inspect the prototypical problem of deciding whether the majority of bits in the input are 1's. □

We now move on to prove SPACE($o(\log n)$) $\subseteq$ MRC^0. It is worth noting that this is a strictly stronger statement than Theorem 1. That is, REGULAR = SPACE($O(1)$) $\subsetneq$ SPACE($o(\log n)$). Several non-trivial examples of languages that witness the strictness of this containment are given in [19].

The proof is very similar to the proof of Theorem 1: Instead of the processors computing the entire table of state-to-state transitions of a DFA, the processors now compute the entire table of all transitions possible among the configurations of the work tape of a Turing machine that uses $o(\log n)$ space.

Theorem 2. $\text{SPACE}(o(\log n)) \subseteq \text{MRC}^0$.

Proof. Let L be a language in $\text{SPACE}(o(\log n))$ and T a Turing machine recognizing L in polynomial time and $o(\log(n))$ space, with a read/write work tape W. Define the first mapper so that the j^{th} processor has the bits from $j\sqrt{n}$ to $(j+1)\sqrt{n}$. Let $\mathcal{C}$ be the set of all possible configurations of W and let S be the states of T. Since the size of S is independent of the input, we can assume that each processor has the transition function of T stored on it.

Now we define ρ_1 as follows: Each processor j constructs the graph of a function $T_j : \mathcal{C} \times \{L, R\} \times S \to \mathcal{C} \times \{L, R\} \times S$, which simulates T when the read head starts on either the left or right side of the jth $\sqrt{n}$ bits of the input and W is in some configuration from $\mathcal{C}$. It outputs whether the read head leaves the y portion of the read tape on the left side, the right side, or else accepts or rejects. To compute the graph of T_j, processor j simulates T using its transition function, which takes polynomial time.

Next we show that the graph of T_j can be stored on processor j by showing it can be stored in $O(\sqrt{n})$ space. Since W is by assumption size $o(\log n)$, each entry of the table is $o(\log n)$, so there are $2^{o(\log n)}$ possible configurations for the tape symbols. There are also $o(\log n)$ possible positions for the read/write head, and a constant number of states T could be in. Hence $|\mathcal{C}| = 2^{o(\log n)} o(\log n) = o(n^{1/3})$. Then processor j can store the graph of T_j as a table of size $O(n^{1/3})$.

The second map function μ_2 sends each T_j (there are $\sqrt{n}$ of them) to a single processor. Each is size $O(n^{1/3})$, and there are $\sqrt{n}$ of them, so a single processor can store all the tables. Using these tables, the final reduce function can now simulate T from starting state to either the accept or reject state by computing $q = T_k^*(\ldots T_2^*(T_1^*(\emptyset, L, \mathit{initial})))$ for some k, where $\emptyset$ denotes the initial configuration of T, *initial* is the initial state of T, and q is either in the accept or reject state. Note T_j^* is the modification of T_j such that if $T_j(x)$ outputs L, then $T_j^*(x)$ outputs R and vice versa. This is necessary because if the read head leaves the j^{th} $\sqrt{n}$ bits to the right, it enters the $j+1^{\text{th}}$ $\sqrt{n}$ bits from the left, and vice versa. Finally, accept if and only if q is in an accept state.

This algorithm successfully simulates T, which decides L, and only takes a constant number of rounds, proving containment. □

5 Hierarchy theorems

In this section we prove two main results (Theorems 3 and 4) about hierarchies within MRC relating to increases in time and rounds. They imply that allowing MRC machines sufficiently more time or rounds strictly increases the computing power of the machines. The first theorem states that for all α, β there are problems $L \notin \text{MRC}[n^\alpha, n^\beta]$ which can be decided by *constant time* MRC machines when given enough extra rounds.

Theorem 3. *Suppose the ETH holds with constant c. Then for every $\alpha, \beta \in \mathbb{N}$ there exists a $\gamma = O(\alpha + \beta)$ such that*

$$\text{MRC}[n^\gamma, 1] \not\subseteq \text{MRC}[n^\alpha, n^\beta].$$

The second theorem is analogous for time, and says that there are problems $L \notin \text{MRC}[n^\alpha, n^\beta]$ that can be decided by a *one round* MRC machine given enough extra time.

Theorem 4. *Suppose the ETH holds with constant c. Then for every $\alpha, \beta \in \mathbb{N}$ there exists a $\gamma = O(\alpha + \beta)$ such that*

$$\text{MRC}[1, n^\gamma] \not\subseteq \text{MRC}[n^\alpha, n^\beta].$$

As both of these theorems depend on the ETH, we first prove a complexity-theoretic lemma that uses the ETH to give a time-hierarchy within linear space TISP. Recall that TISP is the complexity class defined by simultaneous time and space bounds. The lemma can also be described as a time-space tradeoff. For some $b > a$ we prove the existence of a language that can be decided by a Turing machine with simultaneous $O(n^b)$ time and linear space, but cannot be decided by a Turing machine in time $O(n^a)$ even without any space restrictions. It is widely believed such languages exist for *exponential* time classes (for example, TQBF, the language of true quantified Boolean formulas, is a linear space language which is PSPACE-complete). We ask whether such tradeoffs can be extended to polynomial time classes, and this lemma shows that indeed this is the case.

Lemma 5. *Suppose that the ETH holds with constant c. Then for any positive integer a there exists a positive integer $b > a$ such that*

$$\text{TIME}(n^a) \not\subseteq \text{TISP}(n^b, n).$$

Proof. By the ETH, 3-SAT $\in \text{TISP}(2^n, n) \setminus \text{TIME}(2^{cn})$. Let $b := \lceil \frac{a}{c} \rceil + 2$, $\delta := \frac{1}{2}(\frac{1}{b} + \frac{c}{a})$. Pad 3-SAT with $2^{\delta n}$ zeros and call this language L, i.e. let $L := \{x0^{2^{\delta|x|}} \mid x \in \text{3-SAT}\}$. Let $N := n + 2^{\delta n}$. Then $L \in \text{TISP}(N^b, N)$ since $N^b > 2^n$. On the other hand, assume for contradiction that $L \in \text{TIME}(N^a)$. Then, since $N^a < 2^{cn}$, it follows that 3-SAT $\in \text{TIME}(2^{cn})$, contradicting the ETH. □

There are a few interesting complexity-theoretic remarks about the above proof. First, the starting language does not need to be 3-SAT, as the only assumption we needed was its hypothesized time lower bound. We could relax the assumption to the hypothesis that TQBF, the PSPACE-complete language of true quantified Boolean formulas, requires 2^{cn} time for some $c > 0$, or further still to the following complexity hypothesis.

Hypothesis 6. *There is some language in* $\text{TISP}(2^n, 2^{c'n}) \setminus \text{TIME}(2^{cn})$ *for some* $0 < c' < c < 1$.

Second, since $\text{TISP}(n^a, n) \subseteq \text{TIME}(n^a)$, this conditionally proves the existence of a hierarchy within $\text{TISP}(\text{poly}(n), n)$. We note that finding time hierarchies in fixed-space complexity classes was posed as an open question by [21], and so removing the hypothesis or replacing it with a weaker one is an interesting open problem.

Using this lemma we can prove Theorems 3 and 4. The proof of Theorem 3 depends on the following lemma.

Lemma 7. *For every $\alpha, \beta \in \mathbb{N}$ the following holds:*

$$\text{TISP}(n^\alpha, n) \subseteq \text{MRC}[n^\alpha, 1] \ \subseteq \text{MRC}[n^\alpha, n^\beta] \subseteq \text{TISP}(n^{\alpha+\beta+2}, n^2).$$

Proof. The first inequality follows from a simulation argument similar to the proof of Theorem 2. The MRC machine will simulate the $\text{TISP}(n^\alpha, n)$ machine by making one step per round, with the tape (including the possible extra space needed on the work tape) distributed among the processors. The position of the tape is passed between the processors from round to round. It takes constant time to simulate one step of the $\text{TISP}(n^\alpha, n)$ machine, thus in n^α rounds we can simulate all steps. Also, since the machine uses only linear space, the simulation can be done with $O(\sqrt{n})$ processors using $O(\sqrt{n})$ space each. The second inequality is trivial.

The third inequality is proven as follows. Let $T(n) = n^{\alpha+\beta+2}$. We first show that any language in $\text{MRC}[n^\alpha, n^\beta]$ can be simulated in time $O(T(n))$, i.e. $\text{MRC}[n^\alpha, n^\beta] \subseteq \text{TIME}(T(n))$. The r-th round is simulated by applying μ_r to each key-value pair in sequence, shuffle-and-sorting the new key-value pairs, and then applying ρ_r to each appropriate group of key-value pairs sequentially. Indeed, $M(m, r, n, -)$ can be simulated naturally by keeping track of m and r, and adding n to the tape at the beginning of the simulation. Each application of μ_r takes $O(n^\beta)$ time, for a total of $O(n^{\beta+1})$ time. Since each mapper outputs no more than $O(n^c)$ keys, and each mapper and reducer is in $\text{SPACE}(O(n^c))$, there are no more than $O(n^2)$ keys to sort. Then shuffle-and-sorting takes $O(n^2 \log n)$ time, and the applications of ρ_r also take $O(n^{\beta+1})$ time. So a round takes $O(n^{\beta+1} + n^2 \log n)$ time. Note that keeping track of m,r, and n takes no more than the above time. So over $O(n^\alpha)$ rounds, the simulation takes $O(n^{\alpha+\beta+1} + n^{\alpha+2} \log(n)) = O(T(n))$ time. □

Now we prove Theorem 3.

Proof. By Lemma 5, there is a language L in $\text{TISP}(n^\gamma, n) \setminus \text{TIME}(n^{\alpha+\beta+2})$ for some γ. By Lemma 7, $L \in \text{MRC}[n^\gamma, 1]$. On the other hand, because $L \notin \text{TIME}(n^{\alpha+\beta+2})$, we have that $L \notin \text{MRC}[n^\alpha, n^\beta]$ since $\text{MRC}[n^\alpha, n^\beta] \subseteq \text{TIME}(n^{\alpha+\beta+2})$. □

Next, we prove Theorem 4 using a padding argument.

Proof. Let $T(n) = n^{\alpha+\beta+2}$ as in Lemma 7. By Lemma 5, there is a γ such that $\text{TISP}(n^\gamma, n) \setminus \text{TIME}(T(n^2))$ is nonempty. Let L be a language from this set. Pad L with n^2 zeros, and call this new language L', i.e. let $L' = \{x0^{|x|^2} \mid x \in L\}$. Let $N = n + n^2$. There is an $\text{MRC}[1, N^\gamma]$ algorithm to decide L': the first mapper discards all the key-value pairs except those in the first n, and sends all remaining pairs to a single reducer. The space consumed by all pairs is $O(n) = O(\sqrt{N})$. This reducer decides L, which is possible since $L \in \text{TISP}(n^\gamma, n)$. We now claim L' is not in $\text{MRC}[N^\alpha, N^\beta]$. If it were, then L' would be in $\text{TIME}(T(N))$. A Turing machine that decides L' in $T(N)$ time can be modified to decide L in $T(N)$ time: pad the input string with n^2 ones and use the decider for L'. This shows L is in $\text{TIME}(T(n^2))$, a contradiction. □

We conclude by noting explicitly that Theorems 3, 4 give proper hierarchies within MRC, and that proving certain stronger hierarchies imply the separation of L and P.

Corollary 8. *Suppose the ETH. For every α, β there exist $\mu > \alpha$ and $\nu > \beta$ such that*

$$\text{MRC}[n^\alpha, n^\beta] \subsetneq \text{MRC}[n^\mu, n^\beta]$$

and

$$\text{MRC}[n^\alpha, n^\beta] \subsetneq \text{MRC}[n^\alpha, n^\nu].$$

Proof. By Theorem 4, there is some $\mu > \alpha$ such that $\text{MRC}[n^\mu, 1] \not\subseteq \text{MRC}[n^\alpha, n^\beta]$. It is immediate that $\text{MRC}[n^\alpha, n^\beta] \subseteq \text{MRC}[n^\mu, n^\beta]$ and $\text{MRC}[n^\mu, 1] \subseteq \text{MRC}[n^\mu, n^\beta]$. So $\text{MRC}[n^\alpha, n^\beta] \neq \text{MRC}[n^\mu, n^\beta]$. The proof of the second claim is similar. □

Corollary 9. *If* $\text{MRC}[\text{poly}(n), 1] \subsetneq \text{MRC}[\text{poly}(n), \text{poly}(n)]$*, then* $\text{SPACE}(\log(n)) \neq \text{P}$.

Proof.

$$\begin{aligned}\text{SPACE}(\log(n)) &\subseteq \text{TISP}(\text{poly}(n), \log n) \subseteq \text{TISP}(\text{poly}(n), n) \subseteq \text{MRC}[\text{poly}(n), 1] \\ &\subseteq \text{MRC}[\text{poly}(n), \text{poly}(n)] \subseteq \text{P}.\end{aligned}$$

The first containment is well known, the third follows from Lemma 7, and the rest are trivial. □

Corollary 9 is interesting because if any of the containments in the proof are shown to be proper, then $\text{SPACE}(\log(n)) \neq \text{P}$. Moreover, if we provide MRC with a polynomial number of rounds, Corollary 9 says that determining whether time provides substantially more power is at least as hard as separating $\text{SPACE}(\log(n))$ from P. On the other hand, it does not rule out the possibility that $\text{MRC}[\text{poly}(n), \text{poly}(n)] = \text{P}$, or even that $\text{MRC}[\text{poly}(n), 1] = \text{P}$.

6 Discussion and open problems

In this paper we established the first general connections between MapReduce and classical complexity classes, and showed the conditional existence of a hierarchy within MapReduce. Our results also apply to variants of MapReduce, most notably Valiant's BSP model.

Our work suggests some natural open problems. How does MapReduce relate to other complexity classes, such as the circuit class uniform AC^0? Can one improve the bounds from Corollary 8 or remove the dependence on Hypothesis 6? Does Lemma 5 imply Hypothesis 6? Can one give explicit hierarchies for space or time alone, e.g. $\text{MRC}[n^\alpha, \text{poly}(n)] \subsetneq \text{MRC}[n^\mu, \text{poly}(n)]$?

We also ask whether $\text{MRC}[\text{poly}(n), \text{poly}(n)] = \text{P}$. In other words, if a problem has an efficient solution, does it have one with using data locality? A negative answer implies $\text{SPACE}(\log(n)) \neq \text{P}$ which is a major open problem in complexity theory, and a positive answer would likely provide new and valuable algorithmic insights. Finally, while we have focused on the relationship between rounds and time, there are also implicit parameters for the amount of (sublinear) space per processor, and the (sublinear) number of processors per round. A natural complexity question is to ask what the relationship between all four parameters are.

Acknowledgements

We thank Howard Karloff and Benjamin Moseley for helpful discussions.

References

[1] Alexandr Andoni, Aleksandar Nikolov, Krzysztof Onak, and Grigory Yaroslavtsev. Parallel algorithms for geometric graph problems. In *STOC*, pages 574–583, 2014.

[2] Paul Beame, Paraschos Koutris, and Dan Suciu. Communication steps for parallel query processing. In *PODS*, pages 273–284, 2013.

[3] Cheng-Tao Chu, Sang Kyun Kim, Yi-An Lin, YuanYuan Yu, Gary R. Bradski, Andrew Y. Ng, and Kunle Olukotun. Map-reduce for machine learning on multicore. In *NIPS*, pages 281–288, 2006.

[4] Jeffrey Dean and Sanjay Ghemawat. Mapreduce: simplified data processing on large clusters. *Commun. ACM*, 51(1):107–113, 2008.

[5] Ahmed K. Farahat, Ahmed Elgohary, Ali Ghodsi, and Mohamed S. Kamel. Distributed column subset selection on mapreduce. In *ICDM*, pages 171–180, 2013.

[6] Jon Feldman, S. Muthukrishnan, Anastasios Sidiropoulos, Clifford Stein, and Zoya Svitkina. On distributing symmetric streaming computations. *ACM Transactions on Algorithms*, 6(4), 2010.

[7] Lance Fortnow. Time-space tradeoffs for satisfiability. *J. Comput. Syst. Sci.*, 60(2):337–353, 2000.

[8] Michael T. Goodrich, Nodari Sitchinava, and Qin Zhang. Sorting, searching, and simulation in the mapreduce framework. In *ISAAC*, pages 374–383, 2011.

[9] Russell Impagliazzo and Ramamohan Paturi. The complexity of k-sat. *2012 IEEE 27th Conference on Computational Complexity*, 0:237, 1999.

[10] Russell Impagliazzo, Ramamohan Paturi, and Francis Zane. Which problems have strongly exponential complexity? *J. Comput. Syst. Sci.*, 63(4):512–530, 2001.

[11] Seny Kamara and Mariana Raykova. Parallel homomorphic encryption. In *Financial Cryptography Workshops*, pages 213–225, 2013.

[12] Howard Karloff, Siddharth Suri, and Sergei Vassilvitskii. A model of computation for mapreduce. In *SODA '10*, pages 938–948, Philadelphia, PA, USA, 2010. Society for Industrial and Applied Mathematics.

[13] Ravi Kumar, Benjamin Moseley, Sergei Vassilvitskii, and Andrea Vattani. Fast greedy algorithms in mapreduce and streaming. In *SPAA '13*, pages 1–10, New York, NY, USA, 2013. ACM.

[14] Daniel Lokshtanov, Dániel Marx, and Saket Saurabh. Lower bounds based on the exponential time hypothesis. *Bulletin of the EATCS*, 105:41–72, 2011.

[15] Matthew Felice Pace. BSP vs mapreduce. In *Proceedings of the International Conference on Computational Science, ICCS 2012, Omaha, Nebraska, USA, 4-6 June, 2012*, pages 246–255, 2012.

[16] Anish Das Sarma, Foto N. Afrati, Semih Salihoglu, and Jeffrey D. Ullman. Upper and lower bounds on the cost of a map-reduce computation. In *PVLDB'13*, pages 277–288. VLDB Endowment, 2013.

[17] J. C. Shepherdson. The reduction of two-way automata to one-way automata. *IBM J. Res. Dev.*, 3(2):198–200, April 1959.

[18] Konstantin Shvachko, Hairong Kuang, Sanjay Radia, and Robert Chansler. The hadoop distributed file system. In Mohammed G. Khatib, Xubin He, and Michael Factor, editors, *MSST*, pages 1–10. IEEE Computer Society, 2010.

[19] A. Szepietowski. *Turing Machines with Sublogarithmic Space*. Ernst Schering Research Foundation Workshops. Springer, 1994.

[20] Leslie G. Valiant. A bridging model for parallel computation. *Commun. ACM*, 33(8):103–111, 1990.

[21] K. Wagner and G. Wechsung. *Computational Complexity*. Mathematics and its Applications. Springer, 1986.

[22] Ryan Williams. Time-space tradeoffs for counting NP solutions modulo integers. *Computational Complexity*, 17(2):179–219, 2008.

Appendix

A Nonuniform MRC

In this section we show that the original MRC definition of [12] allows MRC machines to decide undecidable languages. This of required a polylogarithmic number of rounds, and also allowed completely different MapReduce machines for different input sizes. For simplicity's sake, we will allow a linear number of rounds, and use our notation MRC$[f(n), g(n)]$ to denote an MRC machine that operates in $O(f(n))$ rounds and each processor gets $O(g(n))$ time per round. In particular, we show that nonuniform MRC$[n, \sqrt{n}]$ accepts all unary languages, i.e. languages of the form $L \subseteq \{1^n \mid n \in \mathbb{N}\}$.

Lemma 10. *Let L be a unary language. Then L is in nonuniform* MRC$[n, \sqrt{n}]$.

Proof. We define the mappers and reducers as follows. Let μ_1 distribute the input as contiguous blocks of $\sqrt{n}$ bits, ρ_1 compute the length of its input, μ_2 send the counts to a single processor, and ρ_2 add up the counts, i.e. find $n = |x|$ where x is the input. Now the input data is reduced to one key-value pair $\langle \star, n \rangle$. Then let ρ_i for $i \geq 3$ be the reducer that on input $\langle \star, i-3 \rangle$ accepts if and only if $1^{i-3} \in L$ and otherwise outputs the input. Let μ_i for $i \geq 3$ send the input to a single processor. Then ρ_{n+3} will accept iff x is in L. Note that ρ_1, ρ_2 take $O(\sqrt{n})$ time, and all other mappers and reducers take $O(1)$ time. All mappers and reducers are also in SPACE$(\sqrt{n})$. □

In particular, Lemma 10 implies that nonuniform MRC$[n, \sqrt{n}]$ contains the unary version of the halting problem. A more careful analysis shows all unary languages are even in MRC$[\log n, \sqrt{n}]$, by having ρ_{i+3} check 2^i strings for membership in L.

B Uniform BSP

We define the BSP model of Valiant [20] similarly to MRC, where essentially key-value pairs are replaced with point-to-point messages.

A BSP machine with p processors is a list $(M_1, \ldots, M_p)$ of p Turing machines which on any input, output a list $((j_1, y_1), (j_2, y_2), \ldots, (j_m, y_m))$ of messages to be sent to other processors in the

next round. Specifically, message y_k is sent to prcessor j_k. A BSP machine operates in rounds as follows. In the first round the input is partitioned into equal-sized pieces $x_{1,0}, \ldots, x_{p,0}$ and distributed arbitrarily to the processors. Then for rounds $r = 1, \ldots, R$,

1. Each processor i takes $x_{i,r}$ as input and computes some number s_i of messages $M_i(x_{i,r}) = \{(j_{i,k}, y_{i,k}) : k = 1, \ldots, s_i\}$.
2. Set $x_{i,r+1}$ to be the set of all messages sent to i (as with MRC's shuffle-and-sort, this is not considered part of processor i's runtime).

We say the machine *accepts* a string x if any machine accepts at any point before round R finishes. We now define uniform deterministic BSP analogously to MRC.

Definition 7 (Uniform Deterministic BSP)**.** A language L is said to be in BSP$[f(n), g(n)]$ if there is a constant $0 < c < 1$, an $O(n^c)$-space and $O(g(n))$-time Turing machine $M(p, y)$, and an $R = O(f(n))$, such that for all $x \in \{0,1\}^n$, the following holds: letting $M_i = M(i, -)$, the BSP machine $M = (M_1, M_2, \ldots, M_{n^c})$ accepts x in R rounds if and only if $x \in L$.

Remark 3. As with MRC, we count the size and number of each message as part of the space bound of the machine generating/receiving the messages. Differing slightly from Valiant, we do not provide persistent memory for each processor. Instead we assume that on processor i, any memory cell not containing a message will form a message whose destination is i. This is without loss of generality since we are not concerned with the cost of sending individual messages.